# AER1418: Assignment 3

Geoff Donoghue April 1, 2019

## Part 1. Heat equation with discontinuous conductivity

We consider a one-dimensional heat equation over the unit domain $\Omega \equiv (0,1)$, in the trial space $\mathcal{V} \equiv \{v \in H^1(\Omega) | v(x=0)=0\}$ we seek some solution $u \in \mathcal{V}$ such that

$$\int_0^1 \kappa(x) \frac{dv}{dx}\frac{du}{dx} dx = v(x=1) \quad \forall v \in \mathcal{V},$$

where we have a discontinuous heat conductivity given by

$$\kappa(x) = \begin{cases} 1 & x \in (0,1/2) \\ 2 & x \in (1/2,1). \end{cases}$$

(a) In order to analytically compute the weak solution to this heat conduction problem we will first split the domain in half and integrate by parts to obtain

$$-\int_0^{\frac{1}{2}} v\frac{d^2u}{dx^2}dx + 2v\frac{du}{dx}\Big|_{x=1} - v\frac{du}{dx}\Big|_{x=1/2} - 2\int_{\frac{1}{2}}^1 v\frac{d^2u}{dx^2}dx = v(x=1).$$

If we hope to enforce the above equation in a point-wise sense we must satisfy the following equations

$$\kappa(x)\frac{d^2u}{dx^2} = 0 \quad x \in (0,1), \qquad \frac{du}{dx}\Big|_{x=1} = \frac{1}{2}, \qquad \frac{du}{dx}\Big|_{x=\frac{1}{2}} = 0, \quad u(x=0)=0,$$

where the last equation is a Dirichlet boundary condition enforced by the trial space in the weak form of the problem. Immediately we see that the solution will take the form of a piecewise linear polynomial, by applying the conditions at the left and right boundaries we obtain

$$u = \begin{cases} c_L x & x \in (0,1/2) \\ \frac{1}{2}x + c_R & x \in (1/2,1). \end{cases}$$

To enforce our interface condition (and also to ensure our solution is $\in \mathcal{V}$) we will say that

$$\frac{du}{dx}\Big|_{x=\frac{1}{2}} = 0 \implies \lim_{\epsilon\to 0} u(\frac{1}{2}+\epsilon) = \lim_{\epsilon \to 0} u(\frac{1}{2}-\epsilon).$$

Applying this condition yields $c_R = \dfrac{2c_L - 1}{4}$. We can say that

$$\int_0^1 \kappa(x)\frac{dv}{dx}\frac{du}{dx}dx = 1\int_0^{\frac{1}{2}} c_L\frac{dv}{dx}dx + 2\int_{\frac{1}{2}}^1 \frac{1}{2}\frac{dv}{dx}dx = c_L v(\frac{1}{2}) + v(1) - v(\frac{1}{2}),$$

and note that $c_L = 1$ will satisfy the weak problem $\forall v \in \mathcal{V}$. The solution is therefore given by

$$u = \begin{cases} x & x \in (0,1/2) \\ \frac{1}{2}x + \frac{1}{4} & x \in (1/2,1). \end{cases}$$

Furthermore, since we have a coercive and continuous bilinear form and a continuous linear form, by the Lax-Milgram theorem this solution is unique. For completeness we can verify that our solution is $\in \mathcal{V}$. The only nontrivial part of this involves proving the existence and boundedness of the weak derivative of $u$. It can be shown that the first weak derivative exists, but the second does not so the solution is $\in H^1(\Omega)$, but $\notin H^2\Omega$.

(b) Since we have $\Omega \subset \mathbb{R}$ as a Lipschitz domain, $H_0^1(\Omega) \subset \mathcal{V} \subset H^1(\Omega)$, our bilinear form is coercive and continuous, our linear form is continuous, and $\mathcal{V}_h \equiv \{v \in V | v \in \mathbb{P}^1(K), K \in \mathcal{T}_h^{\text{even}}\} \subset \mathcal{V}$ we can use Céa's lemma

$$\|u - u_h\|_{\mathcal{V}} \leq \frac{\gamma}{\alpha} \inf_{w_h \in \mathcal{V}_h} \|u - w_h\|_{\mathcal{V}}.$$

Where $\gamma$ and $\alpha$ are the continuity and coercivity constants for the bilinear form, respectively. We note that $\|\cdot\|_{\mathcal{V}} = \|\cdot\|_{H^1(\Omega)}$, and that since our analytical solution $u$ can be expressed as an element of our approximation space (i.e. $u \in \mathcal{V}_h$), we can therefore say that

$$\inf_{w_h \in \mathcal{V}_h} \|u - w_h\|_{\mathcal{V}} = 0.$$

Where evaluating the integral to determine the infinimum is very straightforward, and is therefore omitted. As a result of the infinimum of the error being zero in our approximation space, we say that $\|u - u_h\|_{H^1(\Omega)} = 0$.

(c) We now update our approximation space to consist of quadratic piecewise polynomials, and note that $\mathcal{V}_h' \equiv \{v \in \mathcal{V} | v \in \mathbb{P}^2(K), K \in \mathcal{T}_h^{\text{even}}\} \supset \mathcal{V}_h$. Since any approximation in $\mathcal{V}_h$ exists in $\mathcal{V}_h'$ the infinimum is again zero; furthermore, the conditions required to invoke Céa's lemma are again satisfied, and we can again state that $\|u - u_h\|_{H^1(\Omega)} = 0$.

(d) We no longer have $u \in \mathcal{V}_h$, we note that on every element except for the one located in the middle of the domain ($K_{\text{mid}}$) the solution can be approximated exactly and therefore note that

$$\|u - u_h\|_{H^1(\Omega)} = \|u - u_h\|_{H^1(K_{\text{mid}})} \geq |u - u_h|_{H^1(K_{\text{mid}})} \geq \inf_{w_h \in \mathcal{V}_h} |u - w_h|_{H^1(K_{\text{mid}})}.$$

We can deduce the derivative of the best-fit solution to be $\frac{dw_h}{dx} = \frac{3}{4}$ and state that

$$\begin{aligned}\inf_{w_h \in \mathcal{V}_h} |u - w_h|_{H^1(K_{\text{mid}})} &= \inf_{w_h \in \mathcal{V}_h} \left( \int_{K_{\text{mid}}} \left( \frac{du}{dx} - \frac{dw_h}{dx} \right)^2 dx \right)^{1/2} \\ &= \left[ \int_{\frac{1-h}{2}}^{\frac{1}{2}} (1 - \frac{3}{4})^2 dx + \int_{\frac{1}{2}}^{\frac{1+h}{2}} (\frac{1}{2} - \frac{3}{4})^2 dx \right]^{1/2} \\ &= \left[ \frac{x}{16} \bigg|_{\frac{1-h}{2}}^{\frac{1}{2}} + \frac{x}{16} \bigg|_{\frac{1}{2}}^{\frac{1+h}{2}} \right]^{1/2} \\ &= \frac{1}{4} h^{1/2}.\end{aligned}$$

We can therefore state that the smallest $r$ such that

$$\|u - u_h\|_{H^1(\Omega)} \geq Ch^r = \frac{1}{4} h^{1/2}$$

is $r = 1/2$.

(e) Similarly to the previous problem, we now deduce the derivative of the best-fit solution to be $\frac{dw_h}{dx} = mx + b$, with $m = -\frac{1}{2h}$, and $b = \frac{3h+1}{4h}$. We can substitute this into our infinimum expression to obtain

$$\begin{aligned}\inf_{w_h \in \mathcal{V}_h} |u - w_h|_{H^1(K_{\text{mid}})} &= \inf_{w_h \in \mathcal{V}_h} \left( \int_{K_{\text{mid}}} \left( \frac{du}{dx} - \frac{dw_h}{dx} \right)^2 dx \right)^{1/2} \\ &= \left[ \int_{\frac{1-h}{2}}^{\frac{1}{2}} (1 + \frac{x}{2h} - \frac{3h+1}{4h})^2 dx + \int_{\frac{1}{2}}^{\frac{1+h}{2}} (\frac{1}{2} + \frac{x}{2h} - \frac{3h+1}{4h})^2 dx \right]^{1/2} \\ &= \left[ \frac{h}{96} + \frac{h}{96} \right]^{1/2} \\ &= \frac{1}{\sqrt{48}} h^{1/2}\end{aligned}$$

Where I have evaluated the integral using wolfram alpha. We see that our value for $r$ is once again $\frac{1}{2}$. This goes to show that since our solution is not sufficiently regular in $H^{s+1}(\mathcal{T}_h)$. We expect $r \equiv \min\{s, p\}$, and can deduce that $s = \frac{1}{2}$.

## Part 2. Verification: method of manufactured solution

(a) For the $\mathbb{P}^1$ and $\mathbb{P}^2$ finite element approximations we expect that if our solution is smooth our $H^1(\Omega)$ error norm will converge at a rate of $p$. A convergence plot of the error against $h$ is shown in Figue 1 below.

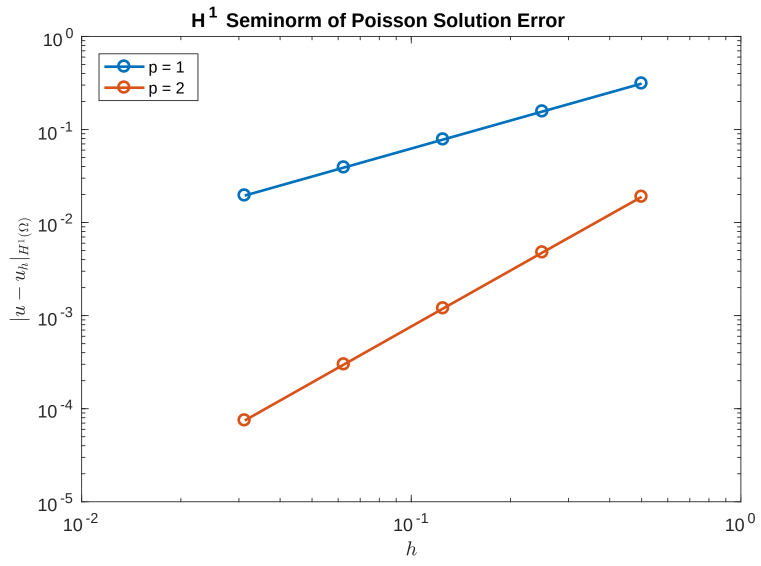

Figure 1: $H^1(\Omega)$ semi norm error convergence for Poisson 1D MMS problem.

The convergence rates were found to be 1.0000 and 1.9998 for the $\mathbb{P}^1$ and $\mathbb{P}^2$ approximations, respectively; these rates agree very well with the theory.

(b) For the $\mathbb{P}^1$ and $\mathbb{P}^2$ finite element approximations we expect that if our solution is smooth our $H^1(\Omega)$ error norm will converge at a rate of $p + 1$. A convergence plot of the error against $h$ is shown in Figure 2 below.

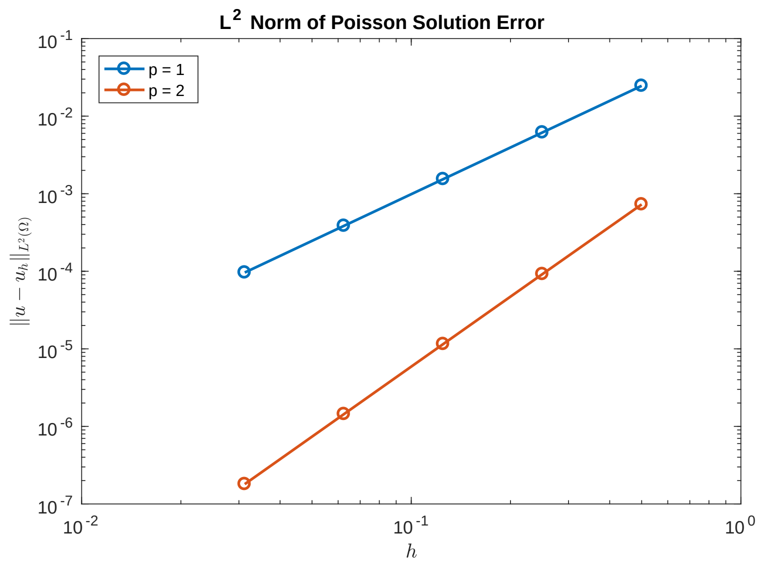

Figure 2: $L^2(\Omega)$ norm error convergence for Poisson 1D MMS problem.

The convergence rates were found to be 2.0000 and 2.9998 for the $\mathbb{P}^1$ and $\mathbb{P}^2$ approximations, respectively; these rates agree very well with the theory.

(c) For the $\mathbb{P}^1$ and $\mathbb{P}^2$ finite element approximations we may naïvely expect that if our solution is smooth our output error $|\ell^{\mathrm{o}}(u) - \ell^{\mathrm{o}}(u_h)|$ will converge at a rate of $2p$. We can however note that since our problem satisfies assumptions 6.1, 6.2, and 6.3 from the notes, and our primal and adjoint solutions are $\in H^1(\Omega) \cap H^{s(')+1}(\mathcal{T}_h)$ we have

$$|\ell^{\mathrm{o}}(u) - \ell^{\mathrm{o}}(u_h)| \lesssim h^{r+r'} |u|_{H^{r+1}(\mathcal{T}_h)} |\psi|_{H^{r'+1}(\mathcal{T}_h)}$$

with $r \equiv \min\{s, p\}$ and $r' \equiv \min\{s', p\}$. Our adjoint equation is given by: find $\psi \in \mathcal{V}$ such that

$$a(w, \psi) = \ell^{\mathrm{o}}(w) \quad \forall w \in \mathcal{V}.$$

We will solve this equation by first converting from the weak form to its strong form by integrating by parts, the weak form of the adjoint problem is

$$w(x=1) \left.\frac{d\psi}{dx}\right|_{x=1} - \cancel{w(x=0)} \left.\frac{d\psi}{dx}\right|_{x=0} = \int_\Omega w(1 + \frac{d^2\psi}{dx^2}) dx.$$

We can deduce form here that the strong form of the problem is

$$-\frac{d^2\psi}{dx^2} = 1, \quad \psi(x=0) = 0, \quad \left.\frac{d\psi}{dx}\right|_{x=1} = 0,$$

the solution of which is

$$\psi = x - \frac{x^2}{2},$$

i.e. a second order polynomial, for which $|\psi|_{H^{r'+1}(\mathcal{T}_h)} = 0$, where $r' = p$.

A convergence plot of the error against $h$ is shown in Figue 3 below.

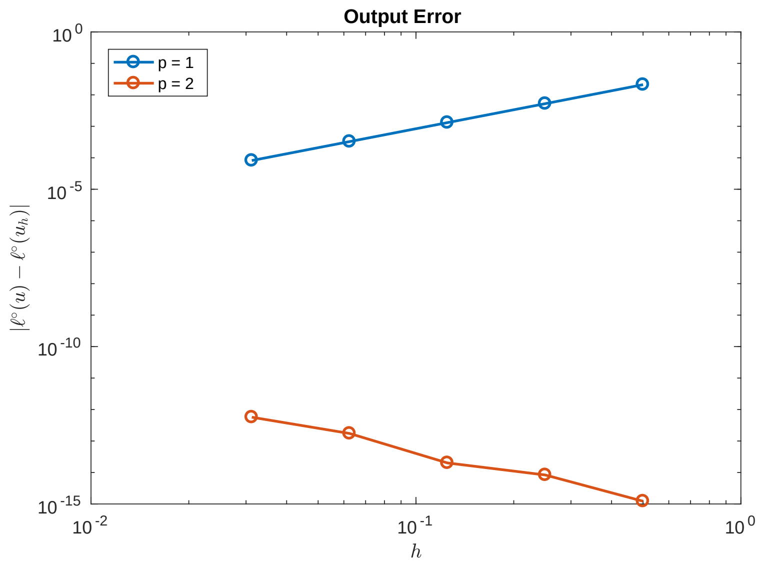

Figure 3: Output error convergence for Poisson 1D MMS problem.

The convergence rates were found to be 2.0001 and $-1.7175$ for the $\mathbb{P}^1$ and $\mathbb{P}^2$ approximations, respectively. The second convergence rate is expected to be zero in the absence of quadrature error (we use $p_{quad} = 4p$), which will be proportional to the number of elements.

## Part 3. Linear Elasticity

We can describe our problem with the following equations

$$-\nabla \cdot \sigma(u) = 0 \text{ in } \Omega, \quad u = u^B \text{ on } \Gamma_D, n \cdot \sigma(u) = g \text{ on } \Gamma_N.$$

We have $\Omega \equiv (0,2) \times (0,1/2)$, $\Gamma_N \equiv \Gamma_{\text{right}} = \{2\} \times (0,1/2)$, $\Gamma_D = \partial\Omega/\Gamma_{\text{right}}$, and $g = [g_{\text{pull}}\ 0]^T$. Multiplying by a test function and integrating by parts yields the weak form of our problem

$$\int_\Omega \nabla v : \sigma(u) dx = \int_{\Gamma_{\text{right}}} v \cdot g ds$$

By substituting in our expression for stress in terms of strain and noting that because $\epsilon(u)$ is symmetric, we have $\nabla v : \epsilon(u) = \epsilon(v) : \epsilon(u)$ we can rewrite our weak form as

$$\int_\Omega 2\mu\epsilon(v) : \epsilon(u) + \lambda \text{tr}(\epsilon(v))(\epsilon(u)) dx = \int_{\Gamma_{\text{right}}} v \cdot g ds.$$

Here we will define our bilinear and linear forms to be respectively

$$a(w,v) \equiv \int_\Omega 2\mu\epsilon(v) : \epsilon(w) + \lambda \text{tr}(\epsilon(v))(\epsilon(w)) dx, \quad \ell(v) \equiv \int_{\Gamma_{\text{right}}} v \cdot g ds.$$

(a) Twice the strain energy density integrated over the domain is

$$\begin{aligned}
\int_\Omega W(u)dx =& 2\int_\Omega \frac{1}{2}\epsilon(u):\sigma(u)dx && \text{(Def. of } W(u))\\
=& \int_\Omega \epsilon(u):[2\mu\epsilon(u)+\lambda \operatorname{tr}(\epsilon(u))I]\,dx && \text{(Def. of } \sigma(u))\\
=& \int_\Omega 2\mu\epsilon(u):\epsilon(u)+\lambda \operatorname{tr}(\epsilon(u))(\epsilon(u))dx\\
=& a(u,u) && \text{(Def. of } a(u,u))\\
=& \ell(u) && \text{(Def. of } u)\\
=& \int_{\Gamma_{\text{right}}} u\cdot g ds && \text{(Def. of } \ell(u))\\
=& \int_{\Gamma_{\text{right}}} u_1 g_{\text{pull}} ds && \text{(Def. of } g(u))\\
=& \ell^{\text{o}}(u) && \text{(Def. of } \ell^{\text{o}}(u)),
\end{aligned}$$

which is exactly our compliance ouput.

(b) Our problem satisfies all the requirements for a minimization formulation with an energy functional

$$u = \arg\min_{w\in\mathcal{V}} J(w), \quad J(v) \equiv \frac{1}{2}a(v,v) - \ell(v) \quad \forall v \in \mathcal{V}.$$

Likewise we have a finite elemenet approximation of the above

$$u_h = \arg\min_{w_h\in\mathcal{V}_h} J(w_h), \quad J(v) \equiv \frac{1}{2}a(v,v) - \ell(v) \quad \forall v \in \mathcal{V}_h.$$

We note that since $\mathcal{V}_h \subset \mathcal{V}$ we can say that

$$J(u) \le J(u_h), \implies \frac{1}{2}a(u,u) - \ell(u) \le \frac{1}{2}a(u_h,u_h) - \ell(u_h).$$

The definitions of $u$ and $u_h$ imply that $a(u,u) = \ell(u)$ and $a(u_h,u_h) = \ell(u_h)$, therefore

$$\frac{1}{2}\ell(u) - \ell(u) \le \frac{1}{2}\ell(u_h) - \ell(u_h).$$

Since we have $\ell(u) = \ell^{\text{o}}(u)$ and $\ell(u_h) = \ell^{\text{o}}(u_h)$ we can say

$$-\frac{1}{2}\ell^{\text{o}}(u) \le -\frac{1}{2}\ell^{\text{o}}(u_h).$$

This implies that

$$\ell^{\text{o}}(u_h) \le \ell^{\text{o}}(u).$$

(c) The code used to solve this problem is shown:

```
function [s, ndof] = plate(h,p,flag)
% PLATE is a driver file for a linear elastic beam problem

% Copyright 2018 Masayuki Yano, University of Toronto

% set equation parameters (plane-stress elasticity i.e. thin plate)
nu = 0.3;
mu = 1/(2*(1+nu));
lambda = nu/(1-nu^2);
```

```
gpull = 0.05;

% set discretization parameters
dim = 2;
if nargin < 2
    p = 2;
end
pquad = 2*p;
if nargin < 1
    h = 0.12;
end

% make reference element
ref = make_ref_tri(p,pquad);

% generate mesh
mesh = make_plate_mesh(h);
if p == 2
    mesh = add_quadratic_nodes(mesh);
end
mesh = make_bgrp(mesh);
if nargin < 3 || flag
    mesh = fix_plate_holes(mesh,ref); % curve hole boundaries
end


% useful variables
[nelem,nshp] = size(mesh.tri);
fprintf('nelem = %d\n',nelem)
nq = length(ref.wq);
nnode = size(mesh.coord,1);

% create local indices to quickly access the local block matrices
ldof{1} = 1:nshp;
ldof{2} = nshp + (1:nshp);

% allocate matrices and vectors
% add storage for other quantites
nldof = 2*nshp;
amat = zeros(nldof,nldof,nelem);
imat = zeros(nldof,nldof,nelem);
jmat = zeros(nldof,nldof,nelem);
fvec = zeros(nldof,nelem);
ivec = zeros(nldof,nelem);

% compute and store local matrices
for elem = 1:nelem
    tril = mesh.tri(elem,:).';

    % compute mesh jacobians
    xl = mesh.coord(tril,:);
    %xq = ref.shp*xl;
    jacq = zeros(nq,dim,dim);
    for j = 1:dim
```

```
            jacq(:,:,j) = ref.shpx(:,:,j)*xl;
        end
        detJq = jacq(:,1,1).*jacq(:,2,2) - jacq(:,1,2).*jacq(:,2,1);
        ijacq = zeros(nq,dim,dim);
        ijacq(:,1,1) =  1./detJq.*jacq(:,2,2);
        ijacq(:,1,2) = -1./detJq.*jacq(:,1,2);
        ijacq(:,2,1) = -1./detJq.*jacq(:,2,1);
        ijacq(:,2,2) =  1./detJq.*jacq(:,1,1);

        % compute quadrature weight
        wqJ = ref.wq.*detJq;

        % compute shape functions
        phixq = zeros(nq,nshp,dim);
        for j = 1:dim
            for k = 1:dim
                phixq(:,:,j) = phixq(:,:,j) + bsxfun(@times,ref.shpx(:,:,k
                   ),ijacq(:,k,j));
            end
        end

        % allocate local matrices
        aaloc = zeros(nldof,nldof);

        % compute local matrices and indices
        % Note: aaloc(ldof{i},ldof{j}) provides access to the (i,j) block
           of
        % the local matrix.
        for i = 1:dim
            for j = 1:dim
                aaloc(ldof{i}, ldof{j}) = aaloc(ldof{i}, ldof{j}) + (i==j)
                   *mu*(phixq(:,:,1)'*diag(wqJ)*phixq(:,:,1) + phixq
                   (:,:,2)'*diag(wqJ)*phixq(:,:,2));
                aaloc(ldof{i}, ldof{j}) = aaloc(ldof{i}, ldof{j}) + mu*
                   phixq(:,:,j)'*diag(wqJ)*phixq(:,:,i);
                aaloc(ldof{i}, ldof{j}) = aaloc(ldof{i}, ldof{j}) + lambda
                   *phixq(:,:,i)'*diag(wqJ)*phixq(:,:,j);
            end
        end

        % insert to global matrices
        amat(:,:,elem) = aaloc;
        imat(:,:,elem) = [repmat(tril,[1,nldof]); nnode + repmat(tril,[1,
           nldof])];
        jmat(:,:,elem) = [repmat(tril',[nldof,1]), repmat(tril',[nldof,1])
           +nnode];
        ivec(:,elem) = [tril; tril+nnode];
    end

    % add boundary contributions
    for bgrp = 1:length(mesh.bgrp)
        for edge = 1:size(mesh.bgrp{bgrp},1)
            % get element, local edge, local nodes, and global nodes
            elem = mesh.bgrp{bgrp}(edge,3);
            ledge = mesh.bgrp{bgrp}(edge,4);
            lnode = ref.f2n(:,ledge);
            tril = mesh.tri(elem,lnode).';

            % compute mesh jacobians
            xl = mesh.coord(tril,:);
            jacq = ref.shpxf*xl;
            detJq = sqrt(sum(jacq.^2,2));

            % compute quadrature weight
            wqJ = ref.wqf.*detJq;

            % compute basis
            phiq = ref.shpf;

            % implement Neumann boundary condition
            if (bgrp == 2)
               ffloc = [phiq'*(wqJ.*gpull); phiq'*(wqJ.*0)];
               fvec([lnode; lnode + nshp],elem) = fvec([lnode; lnode +
                  nshp],elem) + ffloc;
            end
        end
    end

    % assemble matrix
    A = sparse(imat(:),jmat(:),amat(:),dim*nnode,dim*nnode);
    F = accumarray(ivec(:),fvec(:));

    % identify internal and boundary degress of freedom.
    bnodes = [nodes_on_boundary(mesh,ref,1); nnode + nodes_on_boundary(
       mesh,ref,3)];
    inodes = setdiff((1:2*nnode)', bnodes);

    % solve linear system
    U = zeros(nnode,1);
    U(inodes) = A(inodes,inodes)\F(inodes);

    % compute the compliance output
    s = F'*U;
    fprintf('compliance = %14.14f\n', s);
    ndof = nelem*2*nshp;

    % plot strain energy
    if (0)
        figure(4), clf,

        % Loop over elements and compute the strain energy density.
        % Note: the strain energy density should be computed at the
           Lagrange
        % nodes of each element (rather than at the quadrature points)
           such
        % that we can plot the field based on the nodal values.
        E = zeros(nshp,nelem);
        for elem = 1:nelem
        tril = mesh.tri(elem,:).';

        % reference gradient at the Lagrange nodes (not quadrature
           points)
        [~,shpx] = shape_tri(p,ref.xint);

        % compute mesh jacobians at the Lagrange nodes
        xl = mesh.coord(tril,:);
        jacq = zeros(nshp,dim,dim);
        for j = 1:dim
            jacq(:,:,j) = shpx(:,:,j)*xl;
        end
        detJq = jacq(:,1,1).*jacq(:,2,2) - jacq(:,1,2).*jacq(:,2,1);
        ijacq = zeros(nshp,dim,dim);
        ijacq(:,1,1) =  1./detJq.*jacq(:,2,2);
        ijacq(:,1,2) = -1./detJq.*jacq(:,1,2);
        ijacq(:,2,1) = -1./detJq.*jacq(:,2,1);
        ijacq(:,2,2) =  1./detJq.*jacq(:,1,1);

        % compute basis evaluated at the Lagrange nodes
        phixq = zeros(nshp,nshp,dim);
        for j = 1:dim
            for k = 1:dim
                phixq(:,:,j) = phixq(:,:,j) + bsxfun(@times,shpx(:,:,k
                   ),ijacq(:,k,j));
            end
        end

        % compute strain enegy density at the Lagrange nodes
        grad(:,1,1) = phixq(:,:,1)*U(tril);
        grad(:,1,2) = phixq(:,:,2)*U(tril);
        grad(:,2,1) = phixq(:,:,1)*U(tril+nnode);
        grad(:,2,2) = phixq(:,:,2)*U(tril+nnode);
        strain = 1/2 * (grad + permute(grad, [1,3,2]));

        Eloc = mu*(sum(sum(strain.^2,3),2)) + lambda/2 * (strain
           (:,1,1) + strain(:,2,2)).^2;

        E(:,elem) = Eloc;
    end

    % prepare a "new" mesh, mesh2, with the node coordinates modified
    % according to the displacement field U
    mesh2 = mesh;
    mesh2.coord = mesh.coord + reshape(U,[length(mesh.coord),dim]);

    % plot the field
    % Note: The strain energy density field is element-wise
       discontinuous.
    % For E of the size nshp by nelem (as opposed to a vector of
       length
    % nnode), plot_field will plot the discontinuous field.
    plot_field(mesh2,ref,E);
    axis equal;
```

```
        axis([0,2.5,0,0.6]);
        set(gca,'fontsize',16);
    end
    end
```

(d) Figure 4 shows the solution for which the reference output is calculated. It uses 28649 elements ($h = 0.008$) and yields a compliance output of $\ell^o(u_{\text{ref}}) = 0.00456145212$

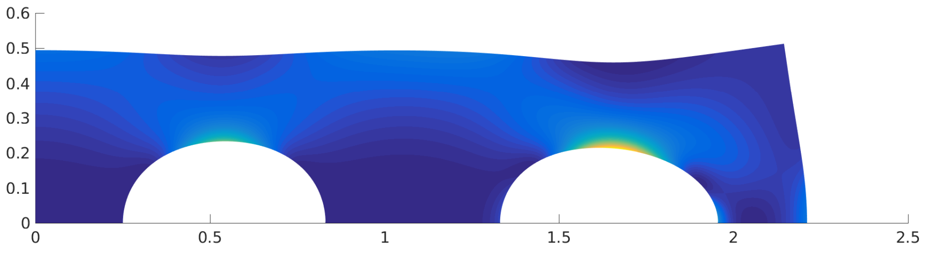

Figure 4: Strain energy density field on the deformed mesh.

(e) Reference outputs for each family of approximations are summarized in Table 1

| | A1 | A2 | A3 |
|---|---|---|---|
| $\ell^o(u_h)$ | 0.0045588902 | 0.0045610541 | 0.0045614521 |

Table 1: Reference compliance outputs, for each approximation family

(f) Figure 5 shows a semilog plot of $\ell^o(u_h)$ against the number of degrees of freedom for each of our approximation families; the reference output is also shown. From part (b) we expect that $\ell^o(u_h) < \ell^o(u_{\text{ref}})$ for each solution.

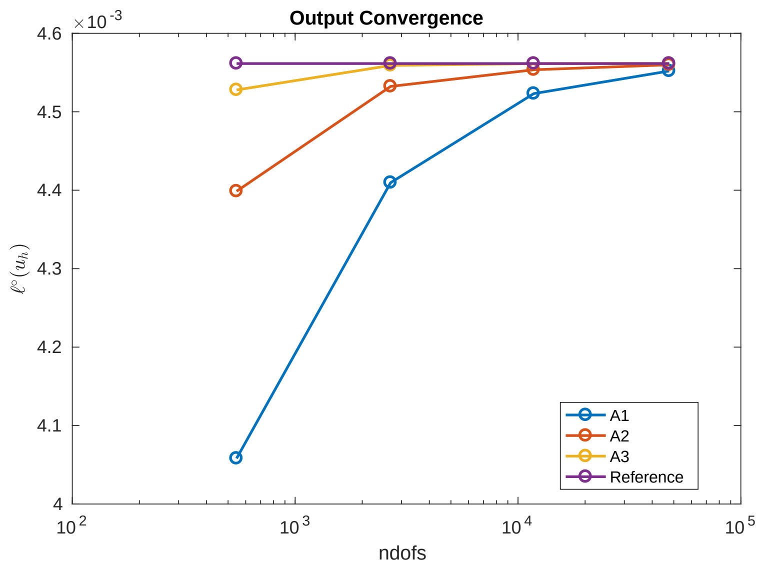

Figure 5: Compliance output for several approximations, note the monotonicity.

We note that as expected, $\ell^o(u_h) < \ell^o(u_{\text{ref}})$, for each solution.

(g) A plot showing a log-log relation between $|\ell^{\mathrm{o}}(u_h) - \ell^{\mathrm{o}}(u_{\mathrm{ref}})|$ against the number of degrees of freedom for each approximation family is shown in Figure 6 below.

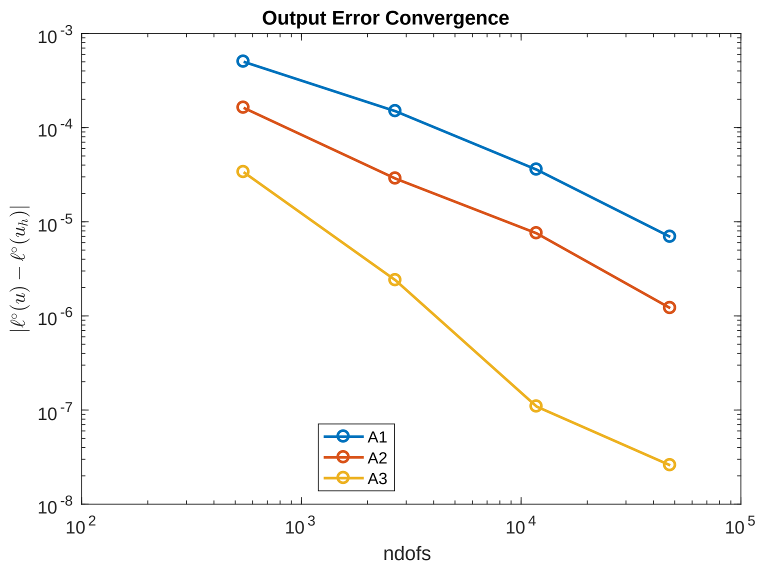

Figure 6: Compliance output error convergence.

(h) The slopes of the above plot are:

| | A1 | A2 | A3 |
|---|---|---|---|
| m | -0.8596 | -1.0001 | -1.8673 |

Table 2: Compliance output error convergence rates with respect to ndofs

We may also compute the convergence rates with respect to $h$, these are:

| | A1 | A2 | A3 |
|---|---|---|---|
| r | 1.9042 | 2.2153 | 4.1364 |

Table 3: Compliance output error convergence rates with respect to $h$

Which are approximately twice the negative convergence rates with respect to the number of degrees of freedom, this is expected as ndofs $\sim 1/h^2$.

Furthermore, we note that if $u \in H^1(\Omega) \cap H^{s+1}(\mathcal{T}_h)$ and $\psi \in H^1(\Omega) \cap H^{s'+1}(\mathcal{T}_h)$ then we have

$$|\ell^{\mathrm{o}}(u_h) - \ell^{\mathrm{o}}(u_{\mathrm{ref}})| \lesssim h^{r+r'} |u|_{H^{r+1}(\mathcal{T}_h)} |\psi|_{H^{r'+1}(\mathcal{T}_h)},$$

where $r \equiv \min\{s, p\}, r' \equiv \min\{s', p\}$. Since we can achieve $r = 2p$ convergence we can conclude that $s > 2$ or that $u \in H^{k \geq 3}(\Omega)$.

## Appendix

Here is additional code, used in part 2; code used to generate refinement plots is simple and therefore not included.

```
function error = compute_output_error(mesh, ref, U)
% Compute absolute value of output error wrt true output
```

```
    % Determine number of elements to loop over
    nelem = size(mesh.tri);

    % Define true output
    output_true = (2*(sqrt(2)+2))/(3*pi);

    % Initialize output, loop over all elements
    output = 0.0;
    for elem = 1:nelem
        % Define preliminaries
        tril = mesh.tri(elem,:).';
        xl = mesh.coord(tril,:);
        jacq = ref.shpx(:,:,1)*xl;
        wqJ = ref.wq.*jacq;

        % Evaluate solution on element
        U_int = ref.shp*U(tril);

        % Integrate to determine output
        output = output + wqJ'*U_int;
    end
    % Return output error
    error = abs(output_true - output);
end

function error = compute_semiH1_error(mesh, ref, U)
% Compute H1 semi norm of error wrt true solution
    % Determine number of elements to loop over
    nelem = size(mesh.tri);

    % Initialize error, loop over all elements
    error = 0.0;
    for elem = 1:nelem
        % Define preliminaries
        tril = mesh.tri(elem,:).';
        xl = mesh.coord(tril,:);
        xq = ref.shp*xl;
        jacq = ref.shpx(:,:,1)*xl;
        wqJ = ref.wq.*jacq;

        % Evaluate true and approximate solution derivatives on element
        U_t = 3*pi/4*cos(3*pi/4*xq);
        U_h = ref.shpx*U(tril)./jacq;

        % Add element wise contribution of error norm
        error = error + (U_t - U_h)'*(wqJ.*(U_t - U_h));
    end
    % Return full norm
    error = sqrt(error);
end

function error = compute_L2_error(mesh, ref, U)
% Compute L2 norm of error wrt true solution
    % Determine number of elements to loop over
    nelem = size(mesh.tri);

    % Initialize error, loop over all elements
    error = 0.0;
    for elem = 1:nelem
        % Define preliminaries
        tril = mesh.tri(elem,:).';
        xl = mesh.coord(tril,:);
        xq = ref.shp*xl;
        jacq = ref.shpx(:,:,1)*xl;
        wqJ = ref.wq.*jacq;

        % Evaluate true and approximate solutions on element
        U_t = sin(3*pi/4*xq);
        U_h = ref.shp*U(tril);

        % Add element wise contribution of error norm
        error = error + (U_t - U_h)'*(wqJ.*(U_t - U_h));
    end
    % Return full norm
    error = sqrt(error);
end
```